BreastCancer

January 3, 2025

[62]:
```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
```

[63]:
```
#Ersteinmal ein Einblick über den Datensatz mittels Panda und der head()␣
 ↪Funktion
import pandas as pd

data_features = pd.DataFrame(data = data.data,
                        columns = data.feature_names)
data_targets = data.target
print(data_features.shape)
print(data_targets.shape)
print(data_features.head().T)
print(data_targets[0:5])
```

```
(569, 30)
(569,)
                              0           1           2           3  \
mean radius               17.990000   20.570000   19.690000   11.420000
mean texture              10.380000   17.770000   21.250000   20.380000
mean perimeter           122.800000  132.900000  130.000000   77.580000
mean area               1001.000000 1326.000000 1203.000000  386.100000
mean smoothness            0.118400    0.084740    0.109600    0.142500
mean compactness           0.277600    0.078640    0.159900    0.283900
mean concavity             0.300100    0.086900    0.197400    0.241400
mean concave points        0.147100    0.070170    0.127900    0.105200
mean symmetry              0.241900    0.181200    0.206900    0.259700
mean fractal dimension     0.078710    0.056670    0.059990    0.097440
radius error               1.095000    0.543500    0.745600    0.495600
texture error              0.905300    0.733900    0.786900    1.156000
perimeter error            8.589000    3.398000    4.585000    3.445000
area error               153.400000   74.080000   94.030000   27.230000
smoothness error           0.006399    0.005225    0.006150    0.009110
compactness error          0.049040    0.013080    0.040060    0.074580
concavity error            0.053730    0.018600    0.038320    0.056610
concave points error       0.015870    0.013400    0.020580    0.018670
symmetry error             0.030030    0.013890    0.022500    0.059630
fractal dimension error    0.006193    0.003532    0.004571    0.009208
```

```
worst radius               25.380000    24.990000    23.570000   14.910000
worst texture              17.330000    23.410000    25.530000   26.500000
worst perimeter           184.600000   158.800000   152.500000   98.870000
worst area               2019.000000  1956.000000  1709.000000  567.700000
worst smoothness            0.162200     0.123800     0.144400    0.209800
worst compactness           0.665600     0.186600     0.424500    0.866300
worst concavity             0.711900     0.241600     0.450400    0.686900
worst concave points        0.265400     0.186000     0.243000    0.257500
worst symmetry              0.460100     0.275000     0.361300    0.663800
worst fractal dimension     0.118900     0.089020     0.087580    0.173000

                                   4
mean radius                20.290000
mean texture               14.340000
mean perimeter            135.100000
mean area                1297.000000
mean smoothness             0.100300
mean compactness            0.132800
mean concavity              0.198000
mean concave points         0.104300
mean symmetry               0.180900
mean fractal dimension      0.058830
radius error                0.757200
texture error               0.781300
perimeter error             5.438000
area error                 94.440000
smoothness error            0.011490
compactness error           0.024610
concavity error             0.056880
concave points error        0.018850
symmetry error              0.017560
fractal dimension error     0.005115
worst radius               22.540000
worst texture              16.670000
worst perimeter           152.200000
worst area               1575.000000
worst smoothness            0.137400
worst compactness           0.205000
worst concavity             0.400000
worst concave points        0.162500
worst symmetry              0.236400
worst fractal dimension     0.076780
[0 0 0 0 0]
```

[124]:
```
import math
#Es gibt also 569 Einträger für 30 Columns, nun zur Frage welche Columns es␣
 ↪gibt:
```

```
column_names = data_features.columns.tolist()
print("Column names:", column_names)
#Auch die Frage was unser Target bedeutet:
print(data.target_names)
print(data_features.iloc[0, 1])
#einen dataframe mit allem
data_set= pd.DataFrame(data_features)

data_set['Diagnose']=data.target

print(data_features.iloc[505,14])
print(data_features[column_names[14]].mean())


print(math.sqrt(data_features[column_names[14]].var()))
```

```
Column names: ['mean radius', 'mean texture', 'mean perimeter', 'mean area',
'mean smoothness', 'mean compactness', 'mean concavity', 'mean concave points',
'mean symmetry', 'mean fractal dimension', 'radius error', 'texture error',
'perimeter error', 'area error', 'smoothness error', 'compactness error',
'concavity error', 'concave points error', 'symmetry error', 'fractal dimension
error', 'worst radius', 'worst texture', 'worst perimeter', 'worst area', 'worst
smoothness', 'worst compactness', 'worst concavity', 'worst concave points',
'worst symmetry', 'worst fractal dimension']
['malignant' 'benign']
10.38
0.02177
0.007040978910369069
0.003002517943839067
```

[132]:
```
import math

#Ersteinmal die Frage ob die Daten einheitliche Datentypen haben:
type_aenderungen=[]
for j in range(data_features.shape[1]):
    my_type = type(data_features.iloc[0, j])
    for i in range(data_features.shape[0]):

        if type(data_features.iloc[i, j])== my_type:
            continue
        else:
            type_aenderungen.append([i,j])

if (len(type_aenderungen)>0):
    for entry in type_aenderungen:
        print("Type Aenderung bei:" +entry)
else:
```

```
    print("Es gibt keine Type Aenderungen")
#Nun gilt es zu überprüfen ob es leere Datensätze bzw. ob Daten fehlen oder ob␣
 ↪es unsinnige Einträge gibt:
falsche_eintraege=[]
for j in range(data_features.shape[1]):
    for i in range(data_features.shape[0]):
        if data_features.iloc[i, j]<= 0:
            #Da bei Concavity und allen diesen begriff enthalteten 0 erlaubt␣
 ↪ist,  müssen diese für 0 ausgenommen werden
            if j!=26 and j!=27 and j!=6 and j!=7 and j!=16 and j!=17:
                falsche_eintraege.append([i,j])
if (len(falsche_eintraege)>0):
    for entry in falsche_eintraege:
        print(entry[0])
        print("Falscher eintrag bei: (" +str(entry[0])+","+str(entry[1])+")")
else:
    print("Es gibt keine falschen Einträge")
#testen ob es ausreißer außerhalb der 3-sigma entfernung gibt:
ausreiser=[]
for j in range(data_features.shape[1]):

    mean = data_features[column_names[j]].mean()
    sigma = math.sqrt(data_features[column_names[j]].var())
    for i in range(data_features.shape[0]):
        if (data_features.iloc[i, j] < mean- 3*sigma or data_features.
 ↪iloc[i,j]>mean +3*sigma) :
            ausreiser.append([i,j])
if (len(ausreiser)>0):
    ausreiser_count=[]
    for entry in ausreiser:
        if entry[0] not in ausreiser_count:
            ausreiser_count.append(entry[0])
        print("Ausreißer bei: (" +str(entry[0])+","+str(entry[1])+")")
    print("Anzahl an Ausreißern:",len(ausreiser_count))
else:
    print("Es gibt keine Ausreißer")
```

```
Es gibt keine Type Aenderungen
Es gibt keine falschen Einträge
Ausreißer bei: (82,0)
Ausreißer bei: (180,0)
Ausreißer bei: (212,0)
Ausreißer bei: (352,0)
Ausreißer bei: (461,0)
Ausreißer bei: (219,1)
Ausreißer bei: (232,1)
```

```
Ausreißer bei: (239,1)
Ausreißer bei: (259,1)
Ausreißer bei: (82,2)
Ausreißer bei: (122,2)
Ausreißer bei: (180,2)
Ausreißer bei: (212,2)
Ausreißer bei: (352,2)
Ausreißer bei: (461,2)
Ausreißer bei: (521,2)
Ausreißer bei: (82,3)
Ausreißer bei: (122,3)
Ausreißer bei: (180,3)
Ausreißer bei: (212,3)
Ausreißer bei: (339,3)
Ausreißer bei: (352,3)
Ausreißer bei: (461,3)
Ausreißer bei: (521,3)
Ausreißer bei: (3,4)
Ausreißer bei: (105,4)
Ausreißer bei: (122,4)
Ausreißer bei: (504,4)
Ausreißer bei: (568,4)
Ausreißer bei: (0,5)
Ausreißer bei: (3,5)
Ausreißer bei: (78,5)
Ausreißer bei: (82,5)
Ausreißer bei: (108,5)
Ausreißer bei: (122,5)
Ausreißer bei: (181,5)
Ausreißer bei: (258,5)
Ausreißer bei: (567,5)
Ausreißer bei: (78,6)
Ausreißer bei: (82,6)
Ausreißer bei: (108,6)
Ausreißer bei: (122,6)
Ausreißer bei: (152,6)
Ausreißer bei: (202,6)
Ausreißer bei: (352,6)
Ausreißer bei: (461,6)
Ausreißer bei: (567,6)
Ausreißer bei: (82,7)
Ausreißer bei: (108,7)
Ausreißer bei: (122,7)
Ausreißer bei: (180,7)
Ausreißer bei: (352,7)
Ausreißer bei: (461,7)
Ausreißer bei: (25,8)
Ausreißer bei: (60,8)
```

```
Ausreißer bei: (78,8)
Ausreißer bei: (122,8)
Ausreißer bei: (146,8)
Ausreißer bei: (3,9)
Ausreißer bei: (71,9)
Ausreißer bei: (152,9)
Ausreißer bei: (318,9)
Ausreißer bei: (376,9)
Ausreißer bei: (504,9)
Ausreißer bei: (505,9)
Ausreißer bei: (122,10)
Ausreißer bei: (138,10)
Ausreißer bei: (212,10)
Ausreißer bei: (258,10)
Ausreißer bei: (417,10)
Ausreißer bei: (461,10)
Ausreißer bei: (503,10)
Ausreißer bei: (12,11)
Ausreißer bei: (83,11)
Ausreißer bei: (122,11)
Ausreißer bei: (192,11)
Ausreißer bei: (416,11)
Ausreißer bei: (473,11)
Ausreißer bei: (557,11)
Ausreißer bei: (559,11)
Ausreißer bei: (561,11)
Ausreißer bei: (12,12)
Ausreißer bei: (108,12)
Ausreißer bei: (122,12)
Ausreißer bei: (212,12)
Ausreißer bei: (258,12)
Ausreißer bei: (417,12)
Ausreißer bei: (461,12)
Ausreißer bei: (503,12)
Ausreißer bei: (122,13)
Ausreißer bei: (212,13)
Ausreißer bei: (265,13)
Ausreißer bei: (368,13)
Ausreißer bei: (461,13)
Ausreißer bei: (503,13)
Ausreißer bei: (71,14)
Ausreißer bei: (116,14)
Ausreißer bei: (122,14)
Ausreißer bei: (213,14)
Ausreißer bei: (314,14)
Ausreißer bei: (345,14)
Ausreißer bei: (505,14)
Ausreißer bei: (12,15)
```

```
Ausreißer bei: (42,15)
Ausreißer bei: (68,15)
Ausreißer bei: (71,15)
Ausreißer bei: (108,15)
Ausreißer bei: (122,15)
Ausreißer bei: (152,15)
Ausreißer bei: (176,15)
Ausreißer bei: (190,15)
Ausreißer bei: (213,15)
Ausreißer bei: (288,15)
Ausreißer bei: (290,15)
Ausreißer bei: (68,16)
Ausreißer bei: (112,16)
Ausreißer bei: (122,16)
Ausreißer bei: (152,16)
Ausreißer bei: (213,16)
Ausreißer bei: (376,16)
Ausreißer bei: (12,17)
Ausreißer bei: (68,17)
Ausreißer bei: (152,17)
Ausreißer bei: (213,17)
Ausreißer bei: (288,17)
Ausreißer bei: (389,17)
Ausreißer bei: (3,18)
Ausreißer bei: (42,18)
Ausreißer bei: (78,18)
Ausreißer bei: (119,18)
Ausreißer bei: (122,18)
Ausreißer bei: (138,18)
Ausreißer bei: (146,18)
Ausreißer bei: (190,18)
Ausreißer bei: (212,18)
Ausreißer bei: (314,18)
Ausreißer bei: (351,18)
Ausreißer bei: (12,19)
Ausreißer bei: (71,19)
Ausreißer bei: (112,19)
Ausreißer bei: (151,19)
Ausreißer bei: (152,19)
Ausreißer bei: (176,19)
Ausreißer bei: (213,19)
Ausreißer bei: (290,19)
Ausreißer bei: (376,19)
Ausreißer bei: (388,19)
Ausreißer bei: (180,20)
Ausreißer bei: (236,20)
Ausreißer bei: (265,20)
Ausreißer bei: (352,20)
```

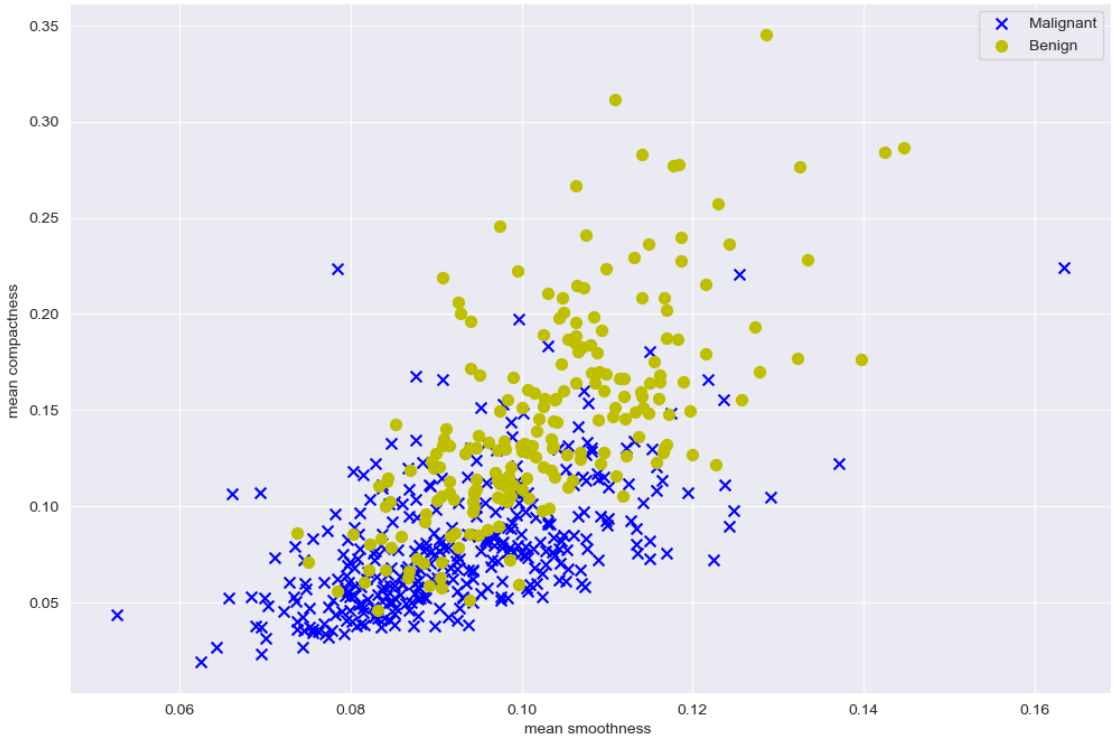
```
Ausreißer bei: (461,20)
Ausreißer bei: (503,20)
Ausreißer bei: (219,21)
Ausreißer bei: (239,21)
Ausreißer bei: (259,21)
Ausreißer bei: (265,21)
Ausreißer bei: (82,22)
Ausreißer bei: (180,22)
Ausreißer bei: (265,22)
Ausreißer bei: (352,22)
Ausreißer bei: (461,22)
Ausreißer bei: (503,22)
Ausreißer bei: (23,23)
Ausreißer bei: (180,23)
Ausreißer bei: (236,23)
Ausreißer bei: (265,23)
Ausreißer bei: (339,23)
Ausreißer bei: (352,23)
Ausreißer bei: (368,23)
Ausreißer bei: (461,23)
Ausreißer bei: (503,23)
Ausreißer bei: (521,23)
Ausreißer bei: (3,24)
Ausreißer bei: (203,24)
Ausreißer bei: (379,24)
Ausreißer bei: (3,25)
Ausreißer bei: (9,25)
Ausreißer bei: (14,25)
Ausreißer bei: (42,25)
Ausreißer bei: (72,25)
Ausreißer bei: (181,25)
Ausreißer bei: (190,25)
Ausreißer bei: (379,25)
Ausreißer bei: (562,25)
Ausreißer bei: (567,25)
Ausreißer bei: (9,26)
Ausreißer bei: (68,26)
Ausreißer bei: (108,26)
Ausreißer bei: (400,26)
Ausreißer bei: (430,26)
Ausreißer bei: (562,26)
Ausreißer bei: (567,26)
Ausreißer bei: (3,28)
Ausreißer bei: (31,28)
Ausreißer bei: (35,28)
Ausreißer bei: (78,28)
Ausreißer bei: (119,28)
Ausreißer bei: (146,28)
```

```
Ausreißer bei: (190,28)
Ausreißer bei: (323,28)
Ausreißer bei: (370,28)
Ausreißer bei: (3,29)
Ausreißer bei: (9,29)
Ausreißer bei: (14,29)
Ausreißer bei: (31,29)
Ausreißer bei: (105,29)
Ausreißer bei: (151,29)
Ausreißer bei: (190,29)
Ausreißer bei: (379,29)
Ausreißer bei: (562,29)
Anzahl an Ausreißern: 74
```

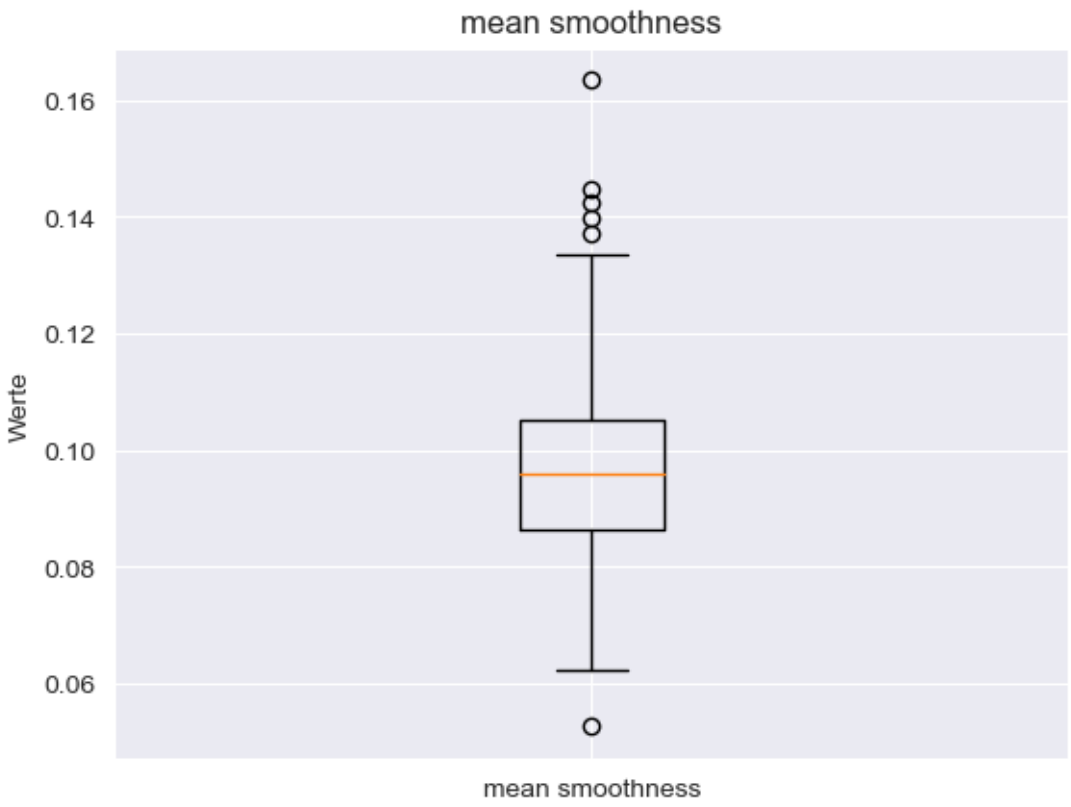
[ ]:
```
#Nun gilt es die Ausreißer zu entfernen
'''
for entry in ausreiser_count:
    data_features.drop(index=entry, inplace=True)
'''
```

[66]:
```
import matplotlib.pyplot as plt
#seperiere die Daten nach Diagnose
data_set_malign = data_set[data_set['Diagnose']==1]
data_set_benign = data_set[data_set['Diagnose']==0]

#50 randomisierte Werte je nach Diagnose getrennt von 2 Indikatoren in einer␣
 ↪2-D Anschauung


fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(data_set_malign['mean smoothness'], data_set_malign['mean␣
 ↪compactness'], s=50, c='b', marker='x', label='Malignant')
ax.scatter(data_set_benign['mean smoothness'], data_set_benign['mean␣
 ↪compactness'], s=50, c='y', marker='o', label='Benign')
ax.legend()
ax.set_xlabel('mean smoothness ')
ax.set_ylabel('mean compactness ')
```

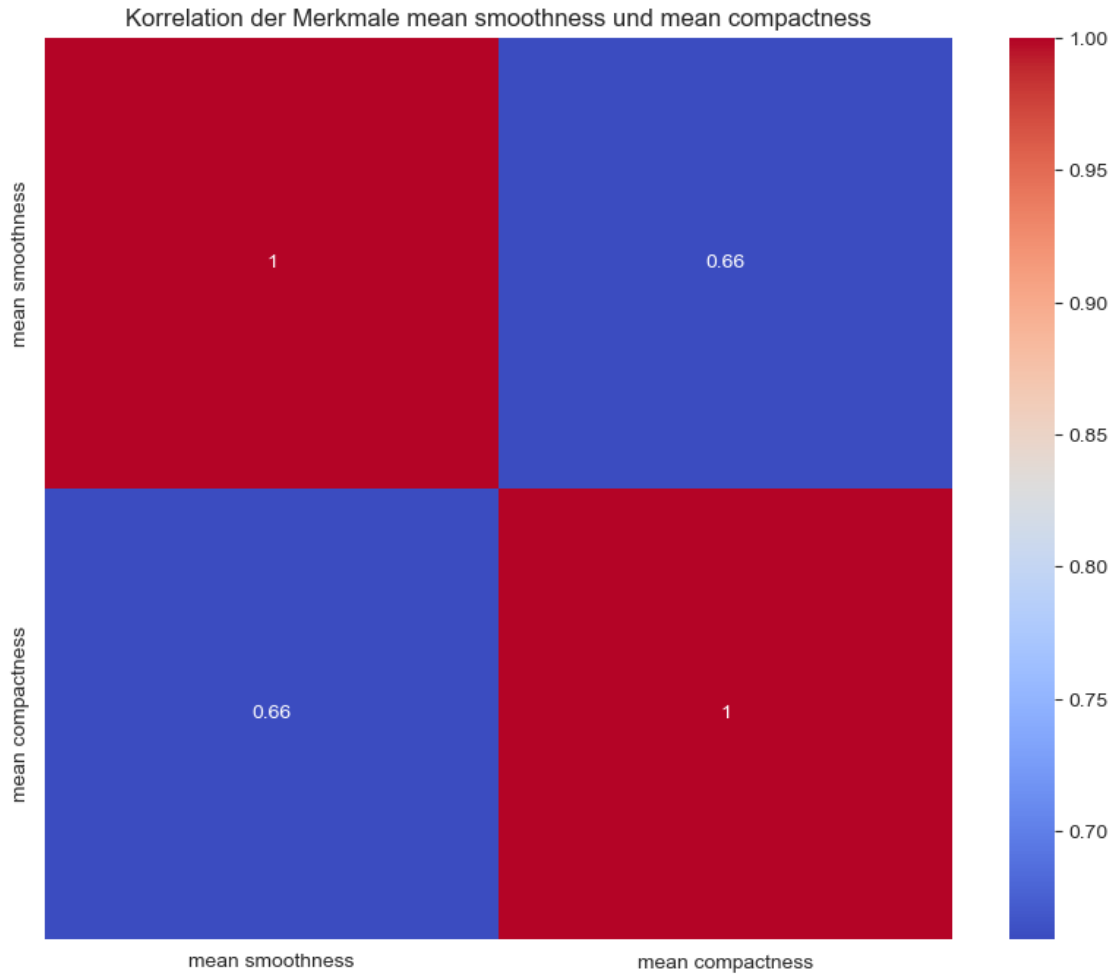
[66]:
```
Text(0, 0.5, 'mean compactness ')
```



[136]:
```
import matplotlib.pyplot as plt
#Mit dem neuen Verstädnis über unseren Datensatz gilt es nun dies zu␣
 ↪visualisieren, um etwaige Ausreißer zu identifizieren
#for i in range(data_features.shape[1]):
i=4
# Boxplot erstellen
plt.boxplot([data_features[column_names[i]]], labels=[column_names[i]])
plt.title(column_names[i])
plt.ylabel('Werte')
plt.show()
```



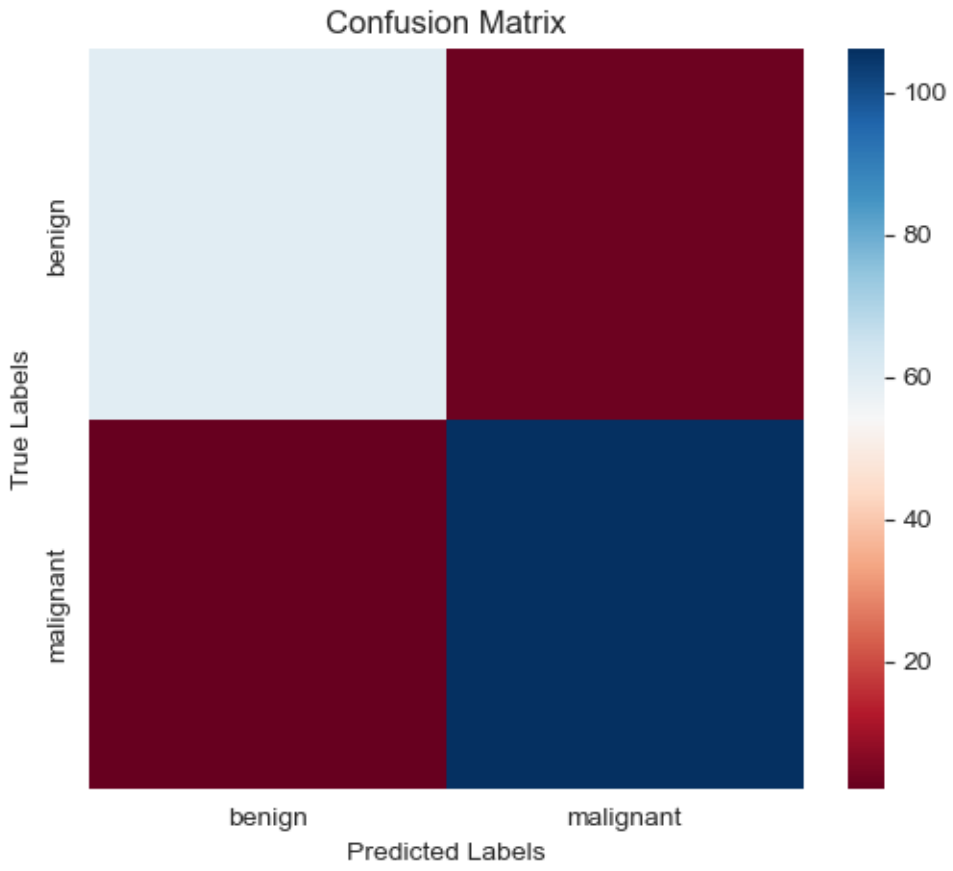
[91]:
```
import seaborn as sns
# Korrelationen mit einer Heatmap visualisiert
plt.figure(figsize=(10, 8))
sns.heatmap(data_features[[column_names[4], column_names[5]]].corr(),␣
 ↪annot=True, cmap="coolwarm")
plt.title("Korrelation der Merkmale " +column_names[4] +" und "+ ␣
 ↪column_names[5] )
plt.show()
```



[69]:
```
from sklearn.metrics import precision_score, roc_auc_score
#Random
from sklearn.model_selection import train_test_split
from sklearn.ensemble import  RandomForestClassifier
from sklearn.metrics import  accuracy_score, confusion_matrix
import seaborn as sns

X = data_features[[column_names[i] for i in range(0, 30)]]

y=data_targets
# Datenaufteilung
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Modellinitialisierung
model_forest =RandomForestClassifier(n_estimators=500, random_state=12)
```

```
# Modelltraining
model_forest.fit(X_train, y_train)

# Vorhersagen
y_pred = model_forest.predict(X_test)


# Bewertung des Modells
accuracy_forest = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy_forest:.4f}")

precision_forest = precision_score(y_test, y_pred)
print(f"Precision: {precision_forest:.4f}")
roc_auc = roc_auc_score(y_test, model_forest.predict_proba(X_test)[:, 1])
print(f"AUC (Area Under the Curve): {roc_auc:.4f}")
import matplotlib.pyplot as plt
plt.figure(figsize=(6, 5))
plt.title("Confusion Matrix")
sns.heatmap(confusion_matrix(y_test, y_pred), cmap="RdBu",␣
 ↪xticklabels=["benign", "malignant"],  yticklabels=["benign", "malignant"])
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

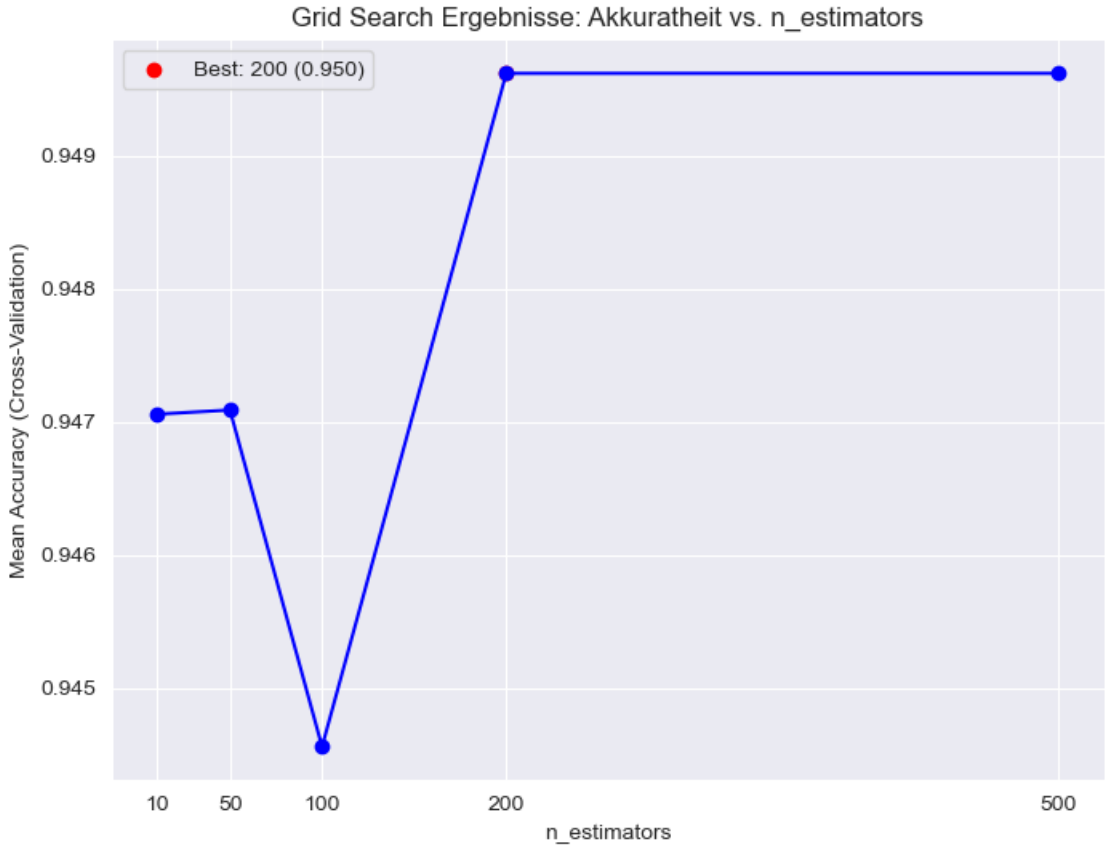
```
Accuracy: 0.9708
Precision: 0.9725
AUC (Area Under the Curve): 0.9966
```



[138]:
```
#gridsearch
from sklearn.model_selection import GridSearchCV

# Parameter Grid für n_estimators
param_grid = {'n_estimators': [10, 50, 100, 200, 500]}
rf_model = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(rf_model, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Beste Parameter und Leistung
print(f"Beste n_estimators: {grid_search.best_params_}")
print(f"Beste Genauigkeit: {grid_search.best_score_}")

results = grid_search.cv_results_
param_values = param_grid['n_estimators']
mean_scores = results['mean_test_score']

# Plot erstellen
plt.figure(figsize=(8, 6))
```

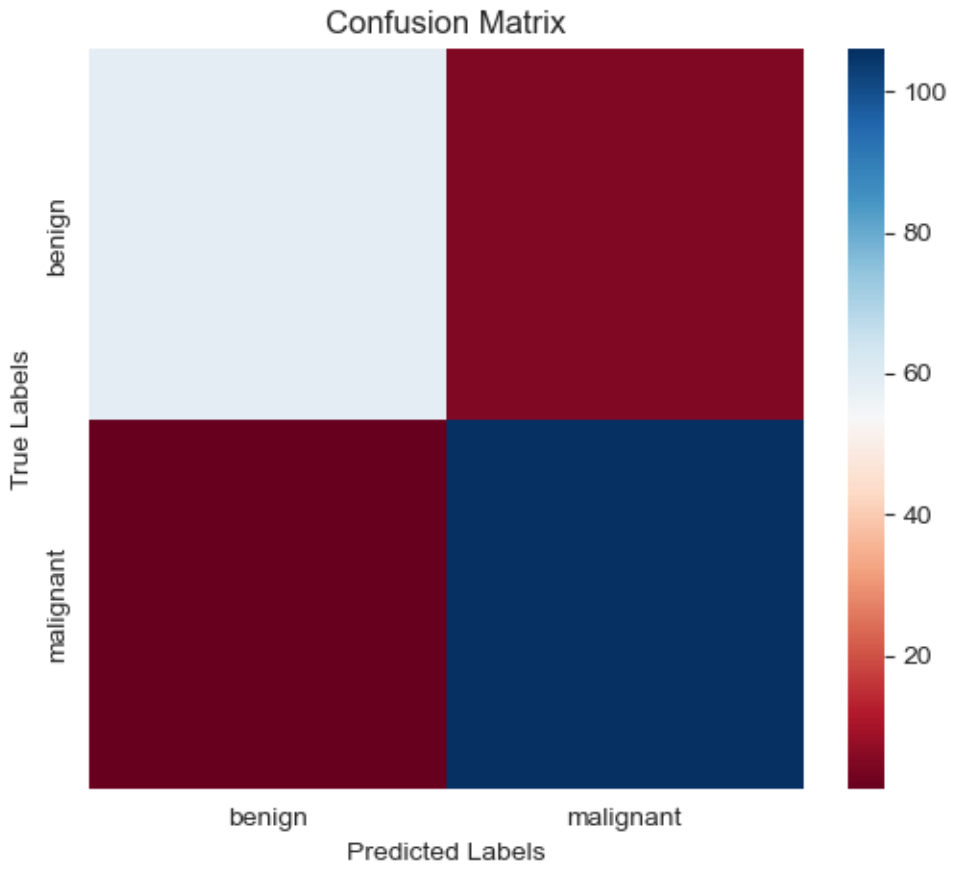
```
plt.plot(param_values, mean_scores, marker='o', linestyle='-', color='blue')
plt.title('Grid Search Ergebnisse: Akkuratheit vs. n_estimators')
plt.xlabel('n_estimators')
plt.ylabel('Mean Accuracy (Cross-Validation)')
plt.xticks(param_values)
plt.grid(True)

# Markieren des besten Werts
best_param = grid_search.best_params_['n_estimators']
best_score = grid_search.best_score_
plt.scatter(best_param, best_score, color='red', label=f'Best: {best_param}␣
 ↪({best_score:.3f})')
plt.legend()

plt.show()
```

```
Beste n_estimators: {'n_estimators': 200}
Beste Genauigkeit: 0.949620253164557
```



[146]:
```
import numpy as np
from sklearn.svm import SVC
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import  accuracy_score, confusion_matrix

X = data_features[[column_names[i] for i in range(0, 30)]]

y=data_targets
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=52)

# Create an SVM model with RBF kernel
svm_model = SVC(kernel='poly', C=10000, random_state=122)

# Fit the model
svm_model.fit(X_train, y_train)

# Predict on the test set
y_pred = svm_model.predict(X_test)


# Evaluate the model
accuracy_svm = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy_svm:.4f}")

precision_svm = precision_score(y_test, y_pred)
print(f"Precision: {precision_svm:.4f}")

import matplotlib.pyplot as plt
plt.figure(figsize=(6, 5))
plt.title("Confusion Matrix")
sns.heatmap(confusion_matrix(y_test, y_pred), cmap="RdBu",␣
 ↪xticklabels=["benign", "malignant"],  yticklabels=["benign", "malignant"])
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

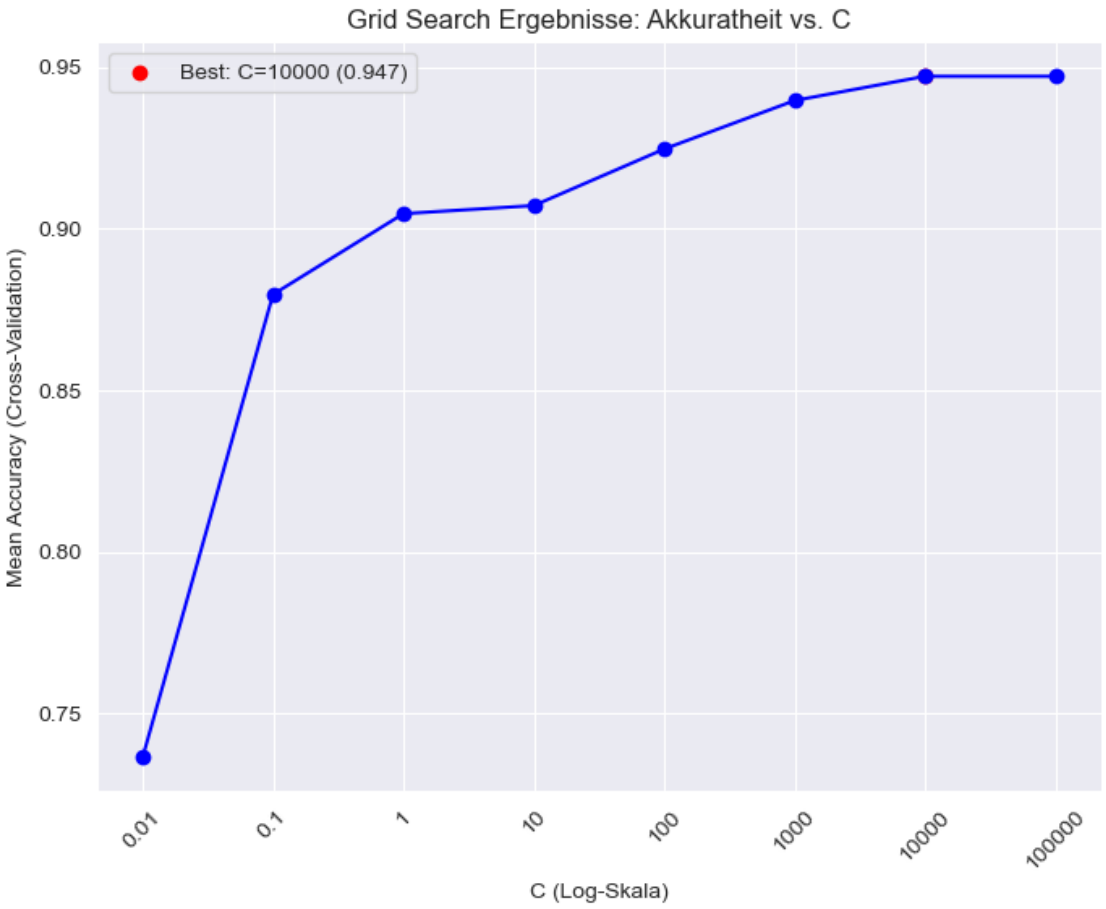
```
Accuracy: 0.9649
Precision: 0.9550
```



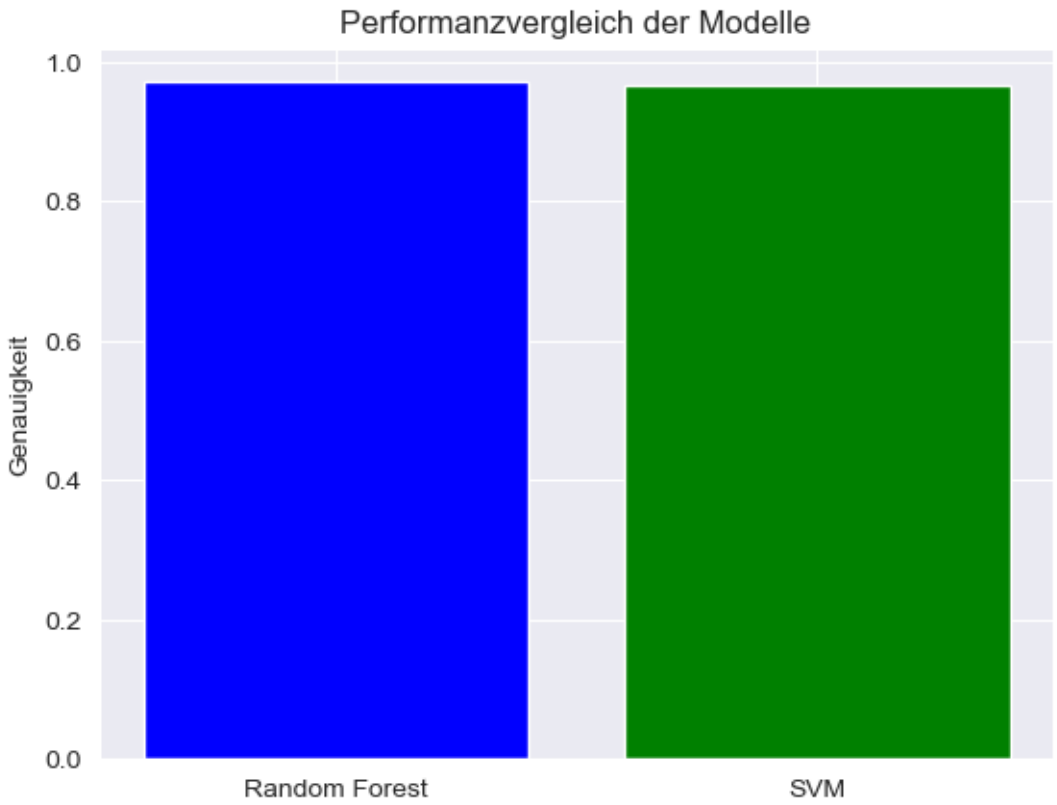
[142]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

# Parameter Grid für C
param_grid = {'C': [.01,0.1, 1, 10, 100, 1000, 10000, 100000]}
svm_model = SVC(kernel='rbf', random_state=42)
grid_search = GridSearchCV(svm_model, param_grid, cv=50)
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=52)
grid_search.fit(X_train, y_train)

# Beste Parameter und Leistung
print(f"Beste C: {grid_search.best_params_}")
print(f"Beste Genauigkeit: {grid_search.best_score_}")

results = grid_search.cv_results_
param_values = param_grid['C']
mean_scores = results['mean_test_score']
```

```python
# Plot erstellen
plt.figure(figsize=(8, 6))
plt.plot(param_values, mean_scores, marker='o', linestyle='-', color='blue')
plt.title('Grid Search Ergebnisse: Akkuratheit vs. C')
plt.xscale('log')
plt.xlabel('C (Log-Skala)')
plt.ylabel('Mean Accuracy (Cross-Validation)')
plt.xticks(param_values, labels=[str(c) for c in param_values], rotation=45)
plt.grid(True)

# Markieren des besten Werts
best_param = grid_search.best_params_['C']
best_score = grid_search.best_score_
plt.scatter(best_param, best_score, color='red', label=f'Best: C={best_param}␣
 ↪({best_score:.3f})')
plt.legend()

plt.show()
```

```
Beste C: {'C': 10000}
Beste Genauigkeit: 0.9471428571428571
```



[80]:
```
#Visualisierung der Genauigkeiten:
plt.bar(['Random Forest', 'SVM'], [accuracy_forest, accuracy_svm],␣
  ↪color=['blue', 'green'])
plt.ylabel("Genauigkeit")
plt.title("Performanzvergleich der Modelle")
plt.show()
```



[86]:
```
# Visualisierung: Balkendiagramm für Accuracy und Precision
labels = ['Accuracy', 'Precision']
forest_scores = [accuracy_forest, precision_forest]
svm_scores = [accuracy_svm, precision_svm]

x = range(len(labels))  # Positionen auf der x-Achse

# Erstelle das Diagramm
fig, ax = plt.subplots(figsize=(8, 6))
# Balken für Accuracy und Precision nebeneinander
bar_width = 0.35  # Breite der Balken
ax.bar(x, svm_scores, width=bar_width, label='SVM', align='center', color='b')
ax.bar([p + bar_width for p in x], forest_scores, width=bar_width,␣
 ↪label='RandomForestClassifier', align='center', color='g')


# Achsenbeschriftungen und Titel
ax.set_xlabel('Modelle')
ax.set_ylabel('Wert')
ax.set_title('Vergleich der Accuracy und Precision von SVM und Random Forest␣
 ↪Classifier')
```

```
ax.set_xticks([p + bar_width / 2 for p in x])
ax.set_xticklabels(labels)
ax.legend()
# Setze den Bereich der y-Achse ab 0.9
ax.set_ylim(0.9, 1.0)

# Zeige das Diagramm
plt.tight_layout()
plt.show()
```